NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1986
Distributed Artificial Intelligence for Communication Network Management

Syracuse University

Robert A. Meyer and Susan E. Conry

This effort was funded partially by the Laboratory Director's fund.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss AFB, NY 13441-5700

[redacted] was reviewed by the RADC Public Affairs Office (PA) and [redacted] to the National Technical Information Service (NTIS). At NTIS [redacted] to the general public, including foreign nations.

[redacted], Vol III (of eight) has been reviewed and is approved for [redacted].

APPROVED: Robert L. Morse

ROBERT L. MORSE, CAPT, USAF
Project Engineer

APPROVED: Bruno Beek

BRUNO BEEK
Technical Director
Directorate of Communications

FOR THE COMMANDER: James W. Hyde III

JAMES W. HYDE, III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DCLD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

ADA197957

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-11, Vol III (of eight)
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC) 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)
6c. ADDRESS (City, State, and ZIP Code) 409 Link Hall Syracuse University Syracuse NY 13244-1240	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) (COES)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F (Over) / PROJECT NO: 5581 / TASK NO: 27 / WORK UNIT ACCESSION NO: 13

11. TITLE (Include Security Classification)
NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1986
Distributed Artificial Intelligence for Communication Network Management

12. PERSONAL AUTHOR(S)
Robert A. Meyer, Susan E. Conry

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM Jan 86 TO Dec 86	June 1988	52

16. SUPPLEMENTARY NOTATION
This effort was performed as a subcontract by Clarkson University to Syracuse University, Office of Sponsored Programs. (See Reverse)

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Distributed Planning, Distributed Artificial Intelligence, Communications Network, Simulation, (Over)
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the second year of the existence of the ~~NAIC~~, on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photo interpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance and explanation system. The specific topic for this volume is the use of knowledge-based systems for communications network management and control via an architecture for a diversely distributed multi-agent system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Arlan L. Morse, Capt, USAF	22b. TELEPHONE (Include Area Code) (315) 330-7751 22c. OFFICE SYMBOL RADC (DCLD)

DD Form 1473, JUN 86 *Previous editions are obsolete*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Item 10. SOURCE OF FUNDING NUMBERS (Continued)

Program Element Number	Project Number	Task Number	Work Unit Number
62702F	4594	18	E2
61101F	LDFP	15	C4
61102F	2304	J5	01
33126F	2155	02	10

Item 18. SUBJECT TERMS (Continued)

Graphical User Interface
Knowledge-based Reasoning

Item 16. SUPPLEMENTARY NOTATION (Continued)

This effort was funded partially by the Laboratory Directors' Fund.

TABLE OF CONTENTS

LIST OF FIGURES

3.1 Introduction

The work described here is based on the completion of the second year of a five year research program designed to answer fundamental questions about the use of knowledge-based systems in communications network management and control. We have developed an architecture for a *diversely distributed, multi-agent system* in which each component is a specialized and localized knowledge-based system designed to provide assistance to the human operator and to cooperate with similar such systems performing other functions and/or located in physically separate facilities. This view of the role of a knowledge-based system as a collection of autonomous, cooperating independent specialists is an important characteristic of our approach to distributed network management.

Modern communications systems, such as the Defense Communications System (DCS), are highly complex collections of equipment and transmission media which currently require, in varying degrees, human intervention for control. The control task is one which requires extensive, specialized knowledge and the ability to reason using this knowledge in solving problems. In the past, system control has been a difficult area to automate because the number of situations which may arise and alternative solutions available are very large, and thus traditional, purely algorithmic approaches have been found lacking.

In our work we have developed a model for communications system management, based on the DCS in Europe. From an in-depth analysis of the problem domain, including field site visits and interviews with operating personnel, we have identified specific problem solving tasks which we believe are suitable for a knowledge-based system. We found three fundamental kinds of knowledge-based problem solving activities required: (1) data interpretation and situation assessment; (2) diagnosis and fault isolation; and (3) planning for resource allocation. In addition to this *functional distribution* of problem solving activities, our model requires a *spatial distribution* of decision making as well. We have designed an architecture to meet these requirements which consists of a distributed knowledge-based system built on a community of problem solving agents. Each agent is a functionally specialized knowledge-based problem solver at a specific site. These agents coordinate and cooperate to solve global problems among themselves, crossing functional or spatial boundaries as required.

An important feature of this architecture is the concept of a *local, shared knowledge base*. Although each problem solving agent has its own, private knowledge about how to perform the specialized problem solving activity for which it is responsible, much of the knowledge needed for problem solving is related to the communication system. This knowledge describes the network structure and organization, the details of different equipment types, and what is known about the current state of the communications system. Since this knowledge is spatially distributed, and shared among the other functional agents at the same local site, it is implemented as a single local knowledge base for each site. An implicit assumption is that each control site only maintains detailed knowledge about the communications equipment and circuits which are within its region of responsibility, and the local knowledge base contains only very limited knowledge about the system outside the local area.

At the present time we have implemented a Distributed AI System (DAISY) testbed which supports simulation of multiple agents on a group of heterogeneous LISP processors. The DAISY testbed incorporates two system building tools which we developed during this effort. SIMULACT is a generic tool for simulating multiple actors in a distributed AI system and is described in section 3.2. In section 3.3 we describe a graphical user interface (GUS) which assists a user in capturing structural knowledge about a communications system. We have also made significant progress in designing a distributed planner for resource allocation. This work has led to the formulation of a new distributed problem solving paradigm we call *Multistage Negotiation*. Section 3.4 illustrates the operation of multistage negotiation with an example problem involving the restoration of communications service following an outage.

3.2 Distributed Simulation Environment

3.2.1 Introduction

In this section we discuss SIMULACT, a tool we have developed for simulating and observing the behavior of networks of distributed systems. Unlike many other systems for simulating distributed environments [2, 6, 19], the goal of our system is not primarily one of achieving speed of execution through parallelism. Instead, it is assumed that the application is inherently distributed, and a natural framework for investigating network behavior is provided. Our system is useful in simulating distributed systems in which processing agents collectively work together towards satisfaction of one or more goals. It is assumed that each agent runs asynchronously and can only communicate with neighboring agents through an exchange of messages. In addition, the activities performed by each agent are assumed to be complex, so that the parallelism is coarse grained.

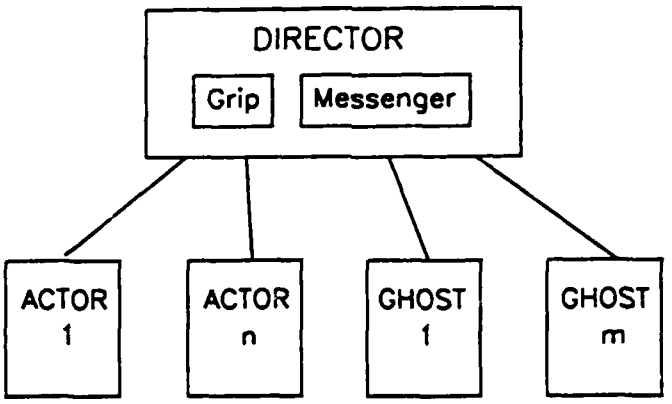
SIMULACT is written in ZetaLisp and extended Common Lisp. It currently runs on the SYMBOLICS 3600 and the TI EXPLORER Lisp machines. Our implementation makes extensive use of flavors to improve data encapsulation and facilitate the modeling of environments in which a group of semiautonomous processes do not share common memory.

3.2.2 System Structure

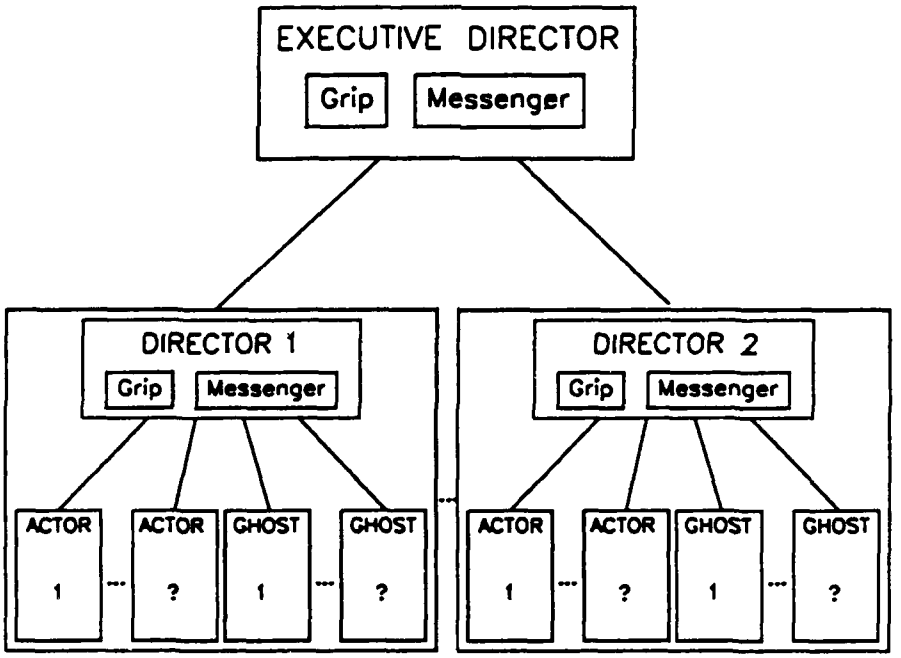
SIMULACT is a distributed time driven simulator capable of achieving improved run time performance through the application of event driven and parallel programming strategies. In this subsection, we describe the major components of SIMULACT at a high level. A discussion of the simulation strategy employed by SIMULACT and some of the implementation details are found in [15].

SIMULACT has a modular structure comprised of four component types: Actors, Ghosts, Directors, and an Executive Director (see Figure 3-1). Actors play the part of processing agents in a distributed system. Multiple instances of agents of the same type in a system are easily incorporated in SIMULACT without requiring any special code development on the part of the user. Ghosts generate information about the environment that naturally occurs in a "real" distributed system and do not represent any physical component of the distributed system being simulated. The Director is responsible for controlling the simulation and all interactions between Actors and Ghosts at each host (there is one Director for each Lisp machine in the network). Finally, the Executive Director coordinates a simulation distributed over a network of Lisp machines. In general, Actors play the role of the entities being simulated, while the Director stays backstage, attempting to get realistic performance from the Actors. In the paragraphs which follow, the role of each of these system components is described in more detail.

Actors in SIMULACT closely resemble the entities which Hewitt calls Actors [12]. Hewitt's Actors are self contained entities which work cooperatively in performing computation and can only be accessed via message passing. Sending and receiving messages are considered to be atomic operations, and messages are accepted one at a time in the order they arrive. Each message, when evaluated, may influence an Actor to



a) Host Level Structure



b) Network Level Structure

Figure 3-1 SIMULACT's Structure

create new Actors, send new communications to other Actors, or to specify the manner in which it will handle new messages.

In SIMULACT, Actors are also self contained and represent the fundamental structure used to simulate concurrency. Our Actors communicate asynchronously by routing messages through the Director. Each Actor in SIMULACT has a "stagename" that is known by its Director and is used in routing these messages. When a Director is asked to transmit a message, it does so by either updating the appropriate Actor's "mailbox" directly, or routing the message to the appropriate Lisp machine through the Executive Director (if the destination agent resides on a different host machine in the network). When an Actor decides to read its mailbox, it has the responsibility of preserving the incoming messages and choosing which one, if any, it will currently respond to. The content and form of messages are independent of SIMULACT, and are determined by the application programmer.

The connectivity of Actors may be fixed or dynamic depending on the physical system being simulated. In a rigidly connected system, each Actor knows the stagenames of Actors with whom it can directly communicate. It is also possible for an Actor to know of the existence of a distant Actor to whom it can indirectly send a message. In simulations where the physical connectivity is allowed to change depending on the current state of the system, SIMULACT provides the capability of generating and managing appropriate stagenames on demand. This is analogous to situations in which Hewitt's Actors spawn new Actors.

One issue which must be addressed in any simulation is the representation of events which occur in the external world and may have impact on the state of the simulation. Examples include external inputs to the simulation from its "global environment" as well as inputs which reflect the "side effects" of the simulated system's activity. Ghosts give the application programmer a facility for injecting these factors into the simulation. For example, a Ghost can use its own event-list to send an Actor a message signaling the occurence of some event at a given time. A Ghost's event-list can easily be altered to investigate the performance of the simulated system in subsequent runs. Ghosts can also be used to inject noise, or wrong information into the simulated system so that issues associated with robustness can be easily investigated.

Actors and Ghosts are referred to as Cast members in SIMULACT, since they are very similar in structure. Each Cast member has a top level function associated with it referred to as a "script function". The script function is written by the application programmer and is invoked by SIMULACT to initiate each Cast member's simulation. Differences between these two cast types are specified (in implementation) through daemons associated with the Actor and Ghost data types. These differences arise from the fact that Actors represent the physical system and must be carefully controlled to achieve a realistic simulation. On the other hand, Ghosts are considered part of the overhead associated with the simulation.

Since SIMULACT is a distributed simulator which runs on a network of Lisp machines, mechanisms for controlling activity among several agents resident on a single

host must be provided. Each host in the network uses a Director to control the activity of Cast members residing at that site, and to route messages to and from these Cast members. Inter-host coordination is managed by the Executive Director.

The Director delegates the task of controlling Cast member activity to the Grip. The job of routing messages to and from Cast members is given to the Messenger. The responsibilities of the Grip range from setting up and initializing each Cast member's local environment to managing and executing the Actor and Ghost queues. The Messenger only deals with the delivery and routing of messages. When a message is sent, it is placed directly into the Messenger's "message-center". During each time frame, the Grip invokes the Messenger to distribute the mail. Whenever the destination stagename is known to the Messenger, the message is placed in the appropriate Cast member's mailbox. Otherwise, it is passed to the Executive Director's Messenger and routed to the appropriate Host.

There is one Executive Director in SIMULACT which coordinates the entire simulation over a distributed network. The Executive Director provides the link between Directors necessary for inter-machine communications, it directs each grip so that synchronization throughout the network is maintained, and it handles the interface between the user and SIMULACT.

3.2.3 Simulation Strategy

In this subsection, we describe the simulation strategy employed by SIMULACT without giving extensive details regarding implementation. In SIMULACT, the components of the physical system being simulated are modeled solely by Actors. Only Actors need to be considered when determining the current "real time" associated with a simulation. For this reason, each instance of an Actor is implemented as a process on a Lisp machine. In general, an Actor's script function is written by the application programmer so that it never "terminates". At any given time, the amount of CPU time spent by the Lisp machine in executing an Actor's script function can be determined. This CPU time, along with the Actor's CPW time (Controlled Program Wait time, or the time spent idle but waiting for response to messages sent), is used in computing the elapsed real time for a given Actor.

A Director is responsible for controlling the simulation on the host machine where it resides. Each Director has two local state variables referred to as **actors** and **ghosts**. The scheduling queue used by SIMULACT in controlling the simulation (relative to each host) is comprised of these two components. The Director advances the simulation locally by one time frame as follows:

(1) When the Executive Director signals the Grip to begin executing a time frame, the Director's current elapsed time is set to the minimum elapsed time of its dependent Actors.

(2) The Grip invokes each Ghost in the ghost queue in a round robin fashion for their current effect on the simulation.

(3) The Messenger distributes all messages present in its message center. Any message with an unknown destination is routed to the Executive Director.

(4) The Grip signals the Executive Director that it has completed step (3) and enters a wait state. This wait state is maintained until all Grips have finished this step.

(5) When the Executive Director signals the Grip to continue, each Actor is removed from the actor queue in a round robin fashion, and
- if it is active, it is allowed to run for one time frame.
- if it is not active, its CPW time is incremented by one time frame.

(6) Go to (1).

From a network perspective the simulation can be viewed as follows:

(1) SIMULACT's current elapsed time has value (n)(time-frame).

(2) The Executive Director sends each Director a message to begin executing a time frame.

(3) The Executive Director collects all inter-machine messages from each Director, and distributes them accordingly.

(4) The Executive Director sends each Director a message to continue executing the current time frame.

(5) SIMULACT's current elapsed time is incremented by one time frame (n=n+1).

(6) Go to (1).

Notice that for each time frame, messages are distributed after all the Ghosts have been invoked. These messages include the current ones just generated by the Ghosts, plus any Actor messages generated during the previous time frame. It is the responsibility of each Cast member to read its mailbox in order to receive these messages, which are tagged with the time of origination. If required, the application programmer can specify a delivery delay time appropriate for the physical system being simulated.

An example of SIMULACT's simulation strategy is depicted in Figure 3-2. The simulation consists of six Actors (A, B, C, D, E, F) distributed over two hosts. During the first time frame, Host 1 allows Actors A, B, and C to run for one time frame each. Likewise, Actors D, E, and F each run for one time frame on Host 2. Neglecting all overhead (including Ghosts) one simulated time frame requires three time frame units to execute. After both machines have completed execution of the current time frame, the next time frame begins.

Allowable time frames in SIMULACT may range from one sixtieth of a second to several seconds. The user specifies his own time frame during system initialization.

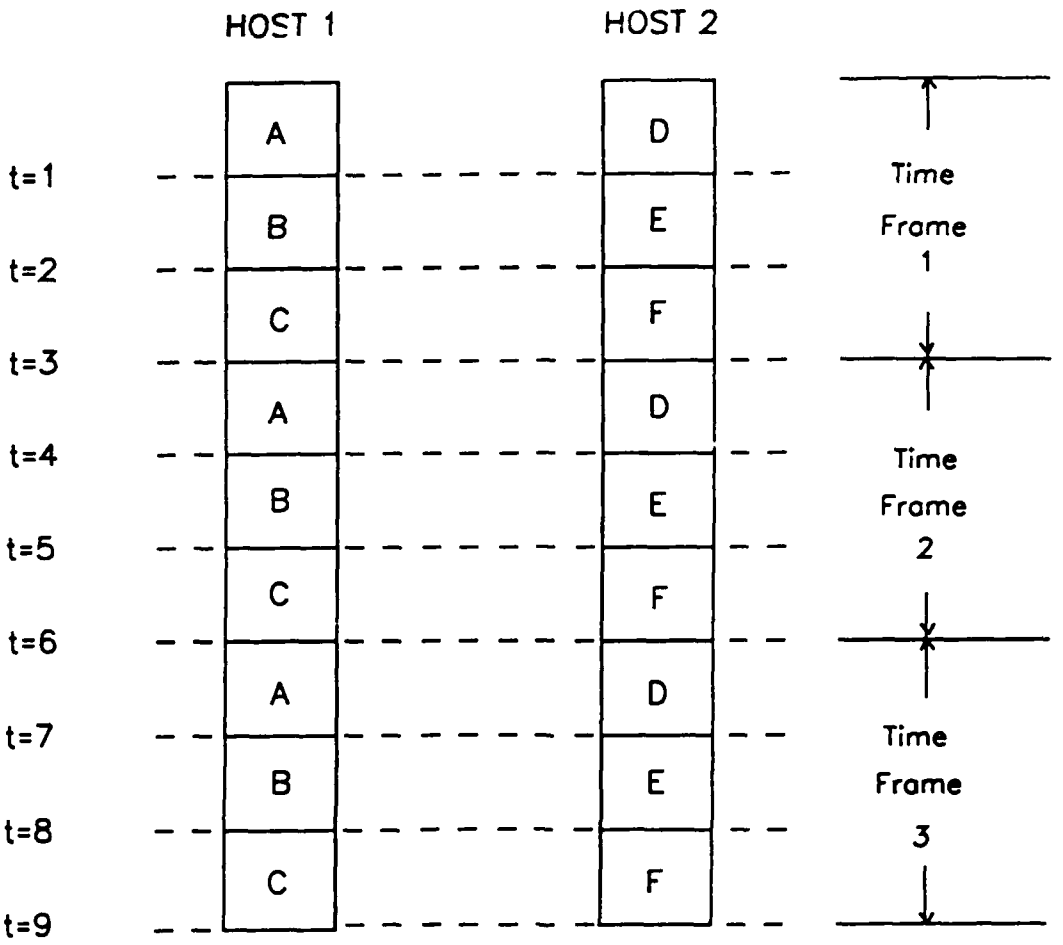


Figure 3-2 Simulation Coordination

3.2.4 SIMULACT'S User Interface

SIMULACT provides the application programmer with a tool for simulating and investigating the behavior of networks of agents. Application code is written for each agent in Lisp as though there were as many machines in the network as agents in the system.

SIMULACT has three initialization modes: New World Generation, Old World Load, and Current World Edit. The generation of a new world in SIMULACT is a menu driven process in which the application programmer specifies the details required to do the simulation. This includes initializing the Executive Director, the Director on each host, and every Cast member. Once the generation of a new world in SIMULACT has been completed, the user has the option of saving it to disk. The Old World Load facility

subsequently loads this information into SIMULACT, without involving the user. The Current World Edit option loads an old world into SIMULACT, and then allows the user to edit it. This is a convenient mechanism for running subsequent simulations with minor alterations.

Initializing the Executive Director configures the Lisp machine network and sets up the user interface. The user must specify the number of hosts involved in the simulation, and set such parameters as the time frame size and the simulated start time. This allows SIMULACT to initialize the gauge facility, which is that part of the user interface which provides control over the simulation (mode select gauge), displays the current simulated time (elapsed time gauge), and allows the application programmer to monitor domain specific characteristics. The initialization of the Executive Director is not complete until all Directors are initialized.

Director initialization is done locally at each host and includes initializing the Cast and their environments. After each Cast member is initialized, the Messenger makes an entry into its "address-book". The address-book is an association list linking a Cast member's stagename to its corresponding Lisp object, and is used to route memos. Director initialization also alters the host machine's operating system to accomodate the specified time frame size.

Each Cast member runs in its own package [22], so it cannot directly access any other Cast member's local state. This ensures that all communications among Cast members are directed through SIMULACT. Each Cast member has one or more initialization files associated with it. These files contain code written by the application programmer describing all activities performed by the Cast member. The Cast member's script function must be contained in this code.

Each Cast member is also given a stagename during initialization. As has been mentioned, these stagenames are used by the Director's Messenger in routing memos. In addition, each cast member is associated with a window pane on the console's screen during initialization. The typical user interface functions (i.e., print, read, etc.) are shadowed so that the user can access each cast member easily through these windows. The collection of all Cast panes makes up the Director's window frame. SIMULACT's interface allows the user to display one Director frame at a time. It also notifies the user when a deexposed frame needs attention.

Since each Cast member maintains its own independent local state, multiple instances of the same Cast type can lead to multiple copies of code. To reduce this costly overhead SIMULACT allows Cast members to use Support packages. A Support package contains code that can be accessed by several Cast members, thus reducing memory requirements. As in any shared memory system, a problem could arise whenever a Support package accesses or alters global information. The underlying assumption concerning independent environments for each Cast member would be violated. To guard against these problems, SIMULACT detects the potential occurrence of improper accesses and warns the user when a Support package tries to instantiate a global variable. Ideally, Support packages should contain purely functional code. However, this restriction would severely restrict the amount of code that can be placed into Support packages.

There are two ways to use Support packages other than for purely functional code. One way is for a Cast member to pass a local data structure as an argument to a Support package function. If that function is "for effect", the result could then be bound appropriately. The other method requires the application programmer to use SIMULACT's "sim-set" function. Basically, the sim-set function allows the Support package to alter a global variable that is present in each of the Cast packages. The goal of the Support package facility is to reduce simulation overhead. Use of support packages does reduce the overhead, but it does so at the expense of requiring that the user have more knowledge about SIMULACT's implementation than is desirable.

3.2.5 Summary

In this section, we have described SIMULACT, a tool we have developed for simulating and observing the behavior of networks of distributed processing agents. This system is currently running as one component in our testbed for investigating problems in distributed artificial intelligence. Our current network configuration contains three Lisp machines.

The system has been particularly useful as a tool in the development of a distributed planning system [3]. It has been used to expose the nature of message traffic in this planner and to develop and debug plan generation in a distributed environment. SIMULACT is also being used as an aid in the development of a distributed diagnosis system. We have found that its modularity and transparency permit us to concentrate on the development of agents which exhibit the desired characteristics rather than on the problems associated with managing the distributed environment.

3.3 Graphical User Interface

3.3.1 Introduction

One important phase in constructing knowledge-based systems is that of knowledge acquisition. The relevant knowledge must be identified, formalized, and represented. Application domains which involve reasoning about physical systems usually include knowledge about the structure of the target system. For this reason, we have designed and implemented a tool to assist an expert in conveying structural knowledge to a machine. Although the present design is directed toward a specific application domain, the design principles employed are domain independent.

When a knowledge engineer questions an expert about problem solving for some physical system, the expert will often begin with a sketch of system components and their interconnections. The symbols used by the expert to represent components and interconnections comprise a language for structural knowledge description. Verbal reasoning and explanation proceed, with the expert using the sketch as an aid in his explanation. It seems clear that a diagram of a physical system often embodies what is known about structure. This knowledge is represented using a set of graphical symbols that are specific to the domain of interest. For this reason, a graphical interface for capturing of structural knowledge should be built upon the symbols used by the expert for structural knowledge description. Part of the gap between expert and machine is bridged by providing a common language.

Structural knowledge of a physical system embodies the components of the system, the behavioral characteristics of these components, component connectivity, and system behavioral characteristics derived from component behavior propagated along connections [1]. Example domains for which structural knowledge is an inherent property include communication networks, automated factory configurations, and electrical circuits. The significance of structural knowledge to our research in distributed problem solving lies in its role in problem solving activities. In the communication network domain, these include fault isolation, service restoral, and performance assessment. Each of these problem solving activities requires structural knowledge to reason about network status and arrive at reasonable solutions.

The design of a graphical interface tool for capturing structural knowledge should not only concern structural details, but also address behavioral characteristics. This is particularly important since the principal problem solving activities in the domain of interest for this task (simulation, fault isolation, service restoral and performance assessment) rely heavily upon knowledge about component and system behavior as well as system structure. Our tool constructs a knowledge base which embodies both structural and behavioral knowledge.

We have developed a Graphical User interface for Structural knowledge (GUS) which provides an interactive, mouse and menu-driven interface for capturing the structural knowledge for a specific application domain. This domain is a large scale communications network system. Our implementation of GUS is running on a Symbolics 3670 Lisp Machine in Zetalisp. A combination of the mouse, menus, window system, and

object-oriented flavors package provided the necessary tools for building GUS. User interaction is primarily via manipulation of a mouse-controlled cursor. Components are selected with the mouse for addition from a library of component icons and then positioned upon the drawing area. Connecting components follows a similar pattern: select the type of connection desired from the library of connection icons and select a component and connect it to another component in the drawing area. Attribute values for objects are easily edited via menus. Continuation of this process results in a complete graphical display representing a communications network with specific equipment configurations. Additionally, a knowledge base which embodies the captured structural knowledge is constructed.

3.3.2 Application Domain Features

Large-scale communications networks form a physical system in which hierarchical structural knowledge is of importance. At the highest level of the structural hierarchy is subregion connectivity. A subregion is comprised of a group of sites, typically spatially clustered, with one site designated as the control center for that particular subregion. Only those links which extend from one subregion to another are considered at this level of abstraction.

Subregion connectivity forms an abstracted view of the network level, the next level of structural detail. The network level represents structure associated with link connectivity among all sites, regardless of subregions. This level is homogeneous in the sense that the only components represented are sites and the only represented connections are links.

The next level of this system hierarchy is the equipment level. Within each site represented at the network level, there is a collection of interconnected equipments which represent a more detailed view of system level connectivity. Equipment types found at this level are radios, multiplexors (MUX), digital patch and access systems (DPAS), and encryption equipment (crypto). At the equipment level, the physical topology is similar to that of the network level, except that the connection media include links as well as supergroups, digroups, jumpers and circuits.

The design of GUS reflects three basic criteria for a knowledge representation language. The first is expressive power. How easily can the expert communicate his knowledge to the system? Our extensive use of domain specific icons representing components and connections provides the requisite expressive power. These icons form a natural vocabulary of symbols which are the foundation of a language for structural knowledge description. The second important criterion is understandability. Can experts understand what the system knows? Machine captured structural knowledge is represented graphically with the same component and connection icons used by the expert to convey structural knowledge to the machine. This commonality of structural knowledge expression supports an environment for natural comprehension of the acquired knowledge. The final criterion is accessibility [8]. Can the system use the knowledge it has captured from the expert? From a "system" perspective, the purpose of this interface is to create a system knowledge base consisting, in part, of structural knowledge. Problem solving agents such as fault isolation, service restoral and performance assessment make

heavy use of structural knowledge represented in the system knowledge base.

3.3.3 Knowledge Base Architecture

The knowledge base is central in effective problem solving activity. The fault isolation, service restoral and performance assessment problem solving agents make extensive use of the knowledge base during their respective problem solving activities. Knowledge represented in the system knowledge base has an important role within each problem solving agent. Network system structural knowledge is necessary during fault isolation techniques in order to trace the equipment of problem areas. Exploitation of the abstracted levels of structural knowledge naturally limits the search space. Service restoral algorithms are dependent upon structural knowledge to determine alternate routes. Performance assessment uses both structural knowledge and state knowledge to accurately interpret a system perspective of performance.

The system knowledge base contains three types of knowledge, as shown in Figure 3-3: graphical knowledge, structural knowledge and state knowledge. Graphical knowledge

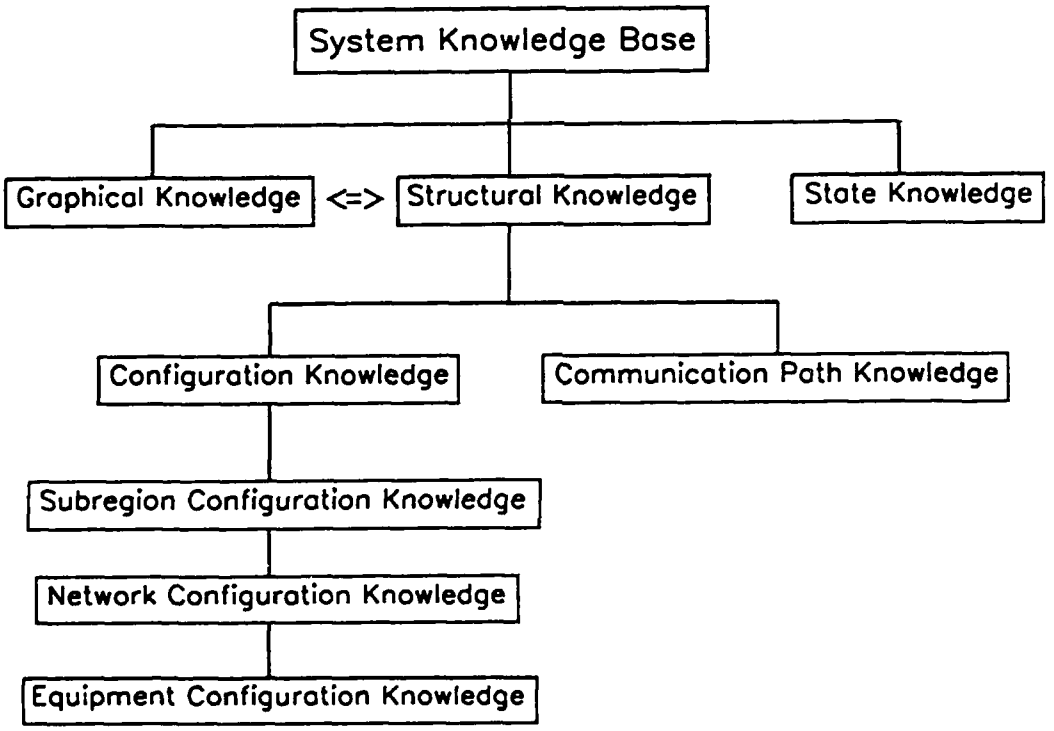


Figure 3-3 Classification of Knowledge Found in the System Knowledge Base

is the primary mechanism for the graphical representation of structural knowledge. Structural knowledge embodies configuration knowledge and communication path knowledge, each of which entails the representation of application domain objects and how they are physically related. State knowledge represents self-descriptive attributes and status of application domain objects. The key point to remember here is that knowledge about structure is common and should be available to each of the different problem solving activities.

As shown in Figure 3-3, there are three levels of configuration knowledge corresponding to the natural hierarchy of application domain structure: subregion, network and equipment configuration knowledge. Knowledge concerning configuration entails specific knowledge of equipment (i.e. connectivity, spatial location), as well as general knowledge such as available status information and expected behavioral characteristics [4]. Consideration of the composite structure of components and connections comprises a topology representing the communication network system as a whole. This network structure forms a natural guideline for search in many instances during the fault isolation problem solving activity. Exploitation of the structural hierarchy permits abstraction of the search space which will expedite fault isolation techniques. Additionally, service restoral activities make use of network structure in the generation of alternate route plans.

Communication path knowledge entails specific combinations of equipment which form a path between two users of a communications system. Such paths are referred to as "circuits" in the equipment editor of GUS. Knowledge is also resident which directs the execution of procedures in the event of user service interruption. Consequently, communication path knowledge is used extensively by fault isolation and service restoral problem solving algorithms.

Graphical knowledge is used for graphical representation of network and equipment configuration knowledge. Graphical knowledge is represented in GUS utilities. The type of information used by the GUS utility to display an item is dependent upon the component or connection being displayed. GUS is able to display all defined component and connection objects.

State knowledge entails all of the attributes of application domain objects which represent object state. For example, which side of a radio is being used for transmitting (A or B), or the status of a supergroup type connection (spare, in use, out of service). Such knowledge is vital during all of the problem solving activities. In addition to operational state knowledge of a specific piece of equipment, knowledge capturing the interpretation of sympathetic alarms or anticipated alarms is also represented.

3.3.4 Design Overview

An object-oriented approach has been chosen for implementation of structural knowledge in the application domain. Such an approach supports a frame-based architecture for the knowledge base. A frame-based representation was chosen to embody the relevant knowledge because this form of knowledge representation easily adapts to

the physical objects of our application domain. The relevant knowledge includes the state knowledge associated with static elements of the application domain and typical communication system event scenarios. Event scenarios are created by the user and provide high-level control over the communications system simulation. Advantages of utilizing a frame-based architecture include easily specified default values, exploitation of inheritance properties, and procedural attachment of knowledge needed in controlling problem solving activities.

From the perspective of integrating graphics capabilities to structural knowledge, the frame-based architecture provided a natural solution to the problems that arise when graphics capabilities must at the same time be both accessible and loosely coupled to the structural knowledge. Such a relationship between graphics capabilities and the structural knowledge base is strongly desired because graphical display knowledge is not desirable nor practical (from an overhead point of view) during problem solving activities.

The first stage in the design involved the specification of type classifications for objects. To conform to the structural theme of representation, four type classifications were formulated: component-objects, connection-objects, utility-objects and window-objects. This is illustrated in Figure 3-4. Component-objects are those

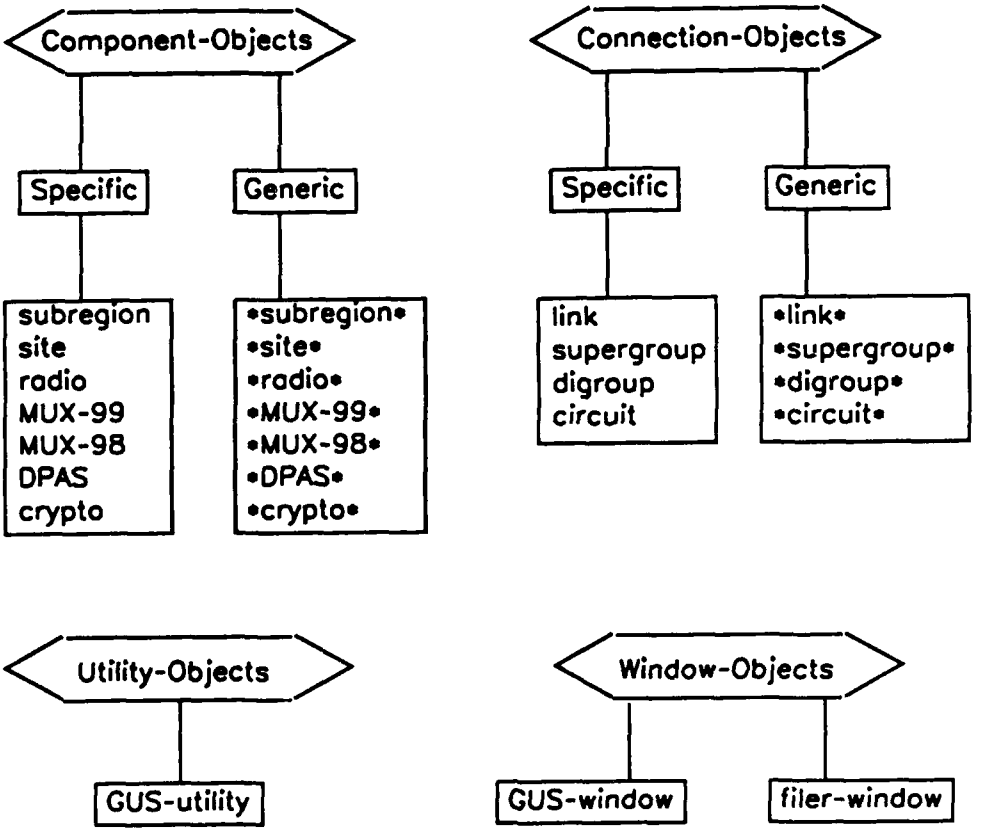


Figure 3-4 Objects and Object Type Classifications

which model components of the application domain. Connection-objects is a collection of objects representing different types of connection media. Utility-objects consists of one object which is responsible for all information regarding graphical input and output. Window-objects contains the window objects used for graphical editors and the file system interface.

Three fundamental functions are provided by GUS. First, structural knowledge is captured from an expert. Second, this knowledge is interpreted and represented in the structural knowledge base. Third, this knowledge is displayed graphically, as indicated in Figure 3-5.

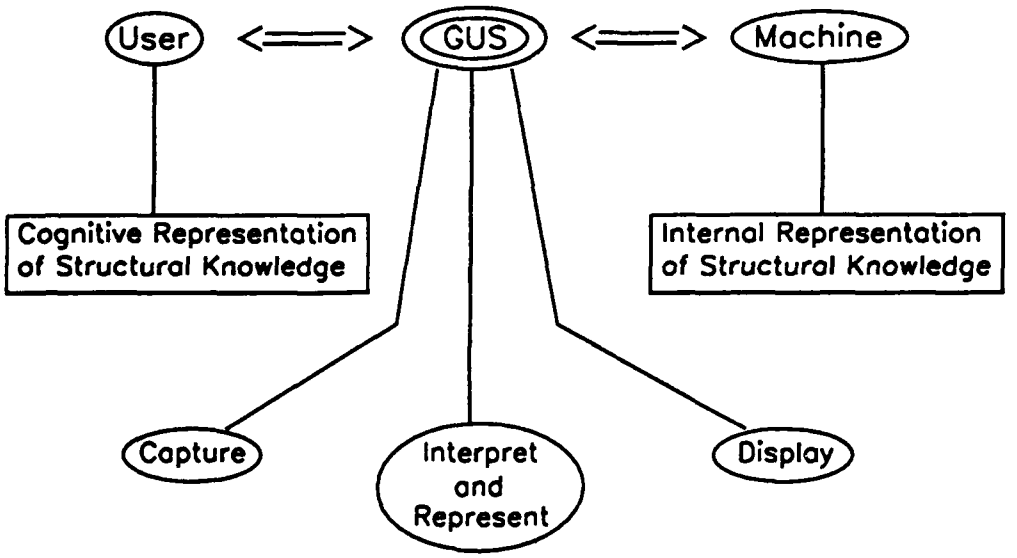


Figure 3-5 Functionality in GUS

Knowledge about the structure of our application domain is captured via interpretation of graphical input. Graphical input is performed by mousing component icons, adding them to a configuration, and selecting connection icons in order to connect added components. As the user draws a component or a connection with the mouse, the system also interprets the graphical addition of this component or connection as an addition to the internal representation of this component or connection to the knowledge base. Graphical addition of components and connections results in the addition of instances of objects representing these components and connections to the knowledge base.

Structural constraints are very important in equipment connectivity. Depending upon the type of connection, certain endpoints are valid and others are not. Therefore, a functional agent tightly coupled with capturing structural knowledge is necessary to guide the user in selecting valid endpoints in the context of the type of connection

being added. With the aid of dynamic mouse sensitivity and highlighting techniques, these connection constraints are effectively enforced.

Upon the completion of a graphical configuration, a knowledge base representing the configuration has been constructed. This knowledge base contains component and connection objects, all of which are related in some physical sense. Two types of knowledge are represented in the knowledge base: graphical and state knowledge. Knowledge concerning graphical display encompasses all information necessary for graphical representation of an object. State knowledge embodies specific and generic attributes of the object. Since a frame-based knowledge representation is employed, both types of knowledge are stored as attribute slot values of appropriate objects.

The third function provided by GUS is the display of structural knowledge. Configurations drawn on the computer screen can be saved and loaded later as needed. A display of the represented knowledge is comprised of the same component and connection icons used in the drawing of the configuration when it was saved. Each component and connection icon is a representation of a component or connection in the knowledge base, respectively. Via mouse manipulation and menus, inspection of icon represented object attributes is enabled.

3.3.5 Design Implementation

This subsection gives more detail about relevant object classes from the perspective of system implementation. In addition, input and output techniques, mouse sensitivity and structural hierarchy from an implementation point of view are discussed.

3.3.5.1 Object Types

As discussed in section 3.3.4, the first stage in an object-oriented design was the specification of object type classifications. The next design step involved the specification of object types within each classification.

Object types of component-objects are site, radio, MUX-99, MUX-98, DPAS, and crypto. Types of connection-objects are link, supergroup, digroup, jumper and circuit. The single object within the utility-objects classification is GUS-utility. The GUS-utility object is the heart of the interface and coordinates graphical display of component-objects and connection-objects.

The window-object classification has two object types: GUS-window and filer-window. GUS-window is a customized window for graphical input and output and is the basis upon which the network and equipment editors are built. User interaction is primarily supported by a level-specific library of icons. For each level of structural detail, there are associated icons which form the basis of user interaction. Filer-window provides an interface window for binary file saving and loading of network configurations.

3.3.5.2 Graphical Input and Output

The input device primarily employed by the interface design is the mouse. The advantage of user input based upon pointing rather than typing a command is that seeing something and pointing to it is significantly easier than typing. From a psychological viewpoint this issue is known as recognition versus recall. Numerous experiments based on distance, target size, and learning found the mouse fastest and with the lowest error rate relative to other input devices such as joysticks and keyboards [16].

Graphical input by the mouse is chiefly supported by two drawing techniques. The first technique is an implementation of a library of items to be displayed as icons on the screen for convenient selection and placement in the drawing. There are two reasons for the use of icons. First, icons are visually more distinctive than a set of words. Second, an icon is able to represent more information than words in a small place, and conservation of screen space is of high priority for items not directly part of a drawing [16].

The second drawing technique implemented is rubber-banding. This technique is utilized during the placement of a connection. It allows the user to strategically place a connection and see what it will look like before fixing it in place. An additional capability provided as part of the connecting process is "tacking down". Tacking permits the user to tack a connection down at specified intermediate points (as opposed to endpoints). This capability enables the user to specify connections other than point-to-point straight lines; specifically, connections comprised of line segments. Therefore, connections can always be specified so that any one segment of any given connection is parallel or perpendicular to other existing connections.

A graphical coding technique is used for graphical output. Icons are used to represent physical component and connection objects of the application domain. The symbols used for component and connection icons of the icon library are the same symbols used for graphical output. For instance, the addition of a radio to an equipment configuration results in the output of a radio symbol to the screen. This radio symbol represents the newly created radio object added to the system knowledge base. This radio symbol is also the same symbol used to comprise the radio icon. Hence, the graphical coding technique stems from the idea of each displayed graphical symbol not only being a visual display, but also a representation of a physical object of the application domain and an object in the knowledge base.

3.3.5.3 Mouse Sensitivity

A common source of erroneous input in a menu-driven graphical interface is using the mouse to select a menu command or displayed object when such an action is out of the current context. For example, choosing to add a connection or remove a component and having all object types displayed be mouse sensitive would be poor design. A solution to such problems is dynamic control of mouse sensitivity to implement the concept of "context sensitive" mouse sensitivity.

By dynamically controlling the mouse sensitivity of displayed objects, we have made the mouse context sensitive in the sense that the items which may be pointed at with the mouse are dependent upon the current context. For example, when an icon command representing a component or connection is selected, a menu of commands will appear. These commands are only associated with the type of object represented by the icon. Component icon commands for "type" component only provide mouse sensitivity for "type" components. Similarly, the "type" connection icon addition command only provides mouse sensitivity for those component objects which are valid "type" connection endpoints.

3.3.5.4 Hierarchical Structural Knowledge

Multi-level structural knowledge is an inherent property of the physical structure of the application domain. This was previously introduced in section 3.3.2 from an application domain perspective. In the following paragraphs, a representational view of multi-level structural knowledge is described. The connectivity of a communications system can be edited at three levels of structural detail, but is only displayed at two (the network and equipment levels). The ability to represent and edit all levels of structural knowledge was a principle design objective.

At the network level, site connectivity is represented by sites and respective link interconnections. Icon symbols representing sites, links, and subregions each have command menus associated with them. By selecting the appropriate icons and subsequent commands, the user assembles a drawing representing site-to-site connectivity of a network. Once the site connectivity has been specified, editing of either the subregion or equipment level of structural detail is permitted.

Subregion editing, the most abstracted level of structural detail, is achieved by the appropriate selection of subregion icon commands. Grouping sites together and designating a control center comprises a subregion and consequently, a new subregion object is added to the system knowledge base.

From the network level, selection of the EQUIPMENT EDITOR icon command and subsequent selection of a site brings the user to the equipment level. At this most detailed level of structure, internal equipment editing and connectivity of a site can be specified. A library of equipment icons, similar in design to the library of network level icons, encapsulates configuration commands. While it is true that the equipment level represents the equipment configuration at a particular site, it is important to remember that equipment configuration is, in a sense, continuous between sites. That is, links specified at this level of detail are representations (pseudo-links) of links existing at the network level. A link at the network level may only be represented at the equipment level if it is connected to the site at which equipment configuration is taking place.

3.3.6 GUS Architectural Design

There are several important features of the design which are discussed in this subsection. First, GUS has two basic editors, the network editor (for specification of

site and subregion connectivity) and the equipment editor (for specification of equipment connectivity). Second, operations are grouped into two categories: components and connections. This decision was made because there are number of logical similarities among operations. This grouping of operations provides an environment in which the user may interact with the system in the same way for all operations in a given category. Similarly, physical objects of the application domain are represented by two groups of objects: component-objects and connection-objects. Third, a variety of connection constraints are enforced during connection specification processes. Each constraint is enforced in a consistant manner by using dynamic mouse sensitivity. Guidance is provided to the user in the form of highlighting to show which displayed objects are mouse sensitive.

Mouse sensitive objects are those displayed objects which react in a controlled manner to the positioning of the mouse cursor over them. Certain terms derived from the word "mouse" are commonly used in the realm of mouse pointing devices and their application. For instance, "mousing" refers to selecting with the mouse, "moused" means "selected", and "mouseable" means the capability to react to the mouse.

3.3.6.1 User Interaction

User interaction with GUS is primarily through mouse and menu driven commands. Mouseable graphic icons and commands in each editor represent groups of type specific sub-commands for the type of object represented by the icon or command. From a graphical perspective, those graphical icons which represent objects that are components of the network model are the same graphical displays used by the graphics support when adding a component to a configuration. This visual one-to-one correspondence is the foundation of our symbolic language for structural knowledge description.

User interaction is implemented by means of two physical devices, the keyboard and the mouse. Keyboard input is limited to situations in which the user must supply data values that cannot be predicted or guessed by the system. The majority of user input is via the mouse, with the use of mouseable commands and graphic icons. Most often the selection of a mouseable command or graphic icon results in presentation of more commands in menu form. The use of menus and pointing devices is a preferred implementation of user interaction because the user is presented a set of possible command choices (dependent upon the current context) rather than being required to remember commands. Additionally, only context dependent commands are available for choice, consequently, erroneous command choices are avoided.

Pop-up menus are used in GUS force user input by remaining displayed until a choice has been made. This type of menu is typically used for selection of available links (link addition at the equipment level) or inputs (choosing a host input number for a connection to a piece of equipment). The item choices displayed at a particular time in one of these menus are the result of some evaluating processes. For example, the input menu only displays spare input numbers of a piece of equipment (those inputs which do not already host a connection).

3.3.6.2 Network and Equipment Level Design

As discussed earlier, our application domain exhibits a high degree of hierarchical structure. GUS captures structural knowledge at two levels of structural detail (the network and equipment levels) via the network editor and the equipment editor. The editing of structure in either editor is limited only by constraints imposed by mouse state diagrams and an implied ordering of data input. In this way, the user is given a freedom of input which is bounded by mouse state.

The network editor is an icon-based menu-driven user interface for the creation, editing and saving of network level components, connections and subregion designations. With simple mouse and menu commands the user can add sites and make link connections by selecting sites as endpoints. The equipment level is similar in design to the network level. Entering the equipment editor places the user in a familiar environment.

The equipment editor is an icon-based menu-driven user interface for the editing of equipment level components and connections. It provides an easy and natural means of configuring equipment within a site location and is accessible only through the network editor by selecting a site at which equipment configuration is desired.

One point of interest, from a human factors perspective, is the specification of connections between equipment. Depending upon the type of connection, certain constraints regarding endpoints must be observed. This was not the case at the network level since there was only one type of connection and one type of component. The only connectivity constraint was the common sense constraint of a link not having the same site for both endpoints.

3.3.6.3 Enforcing Connection Constraints

Types of connections found at the equipment level are links, supergroups, digroups, jumpers, and circuits. As we have mentioned, there are endpoint constraints for each type of connection. In order to enforce connection constraints during the connection specification process, a combination of dynamic mouse sensitivity and graphical highlighting is employed. Highlighting valid choices makes it easy for the user to identify mouse sensitive components. For any connection type the user selects, only those pieces of equipment which are valid endpoints are highlighted and mouse sensitive. Which endpoints are valid is dependent upon the type of connection, which endpoint is being specified, and the presence of spare inputs or outputs to host the connection.

In order to provide design guidelines for connection specification, two general constraints are imposed on the user during the connection specification process. First, a connection will always start at a piece of equipment. Second, if both endpoints of a connection are constrained to be equipment, then the first endpoint will always start at the piece of equipment which hosts the connection as an input.

Link addition or removal at the equipment level does not actually add or remove a link to the knowledge base. Instead, equipment level links can be thought of

conceptually as "pseudo-links" or representations of links at the network level. The network level provides abstracted information concerning network connectivity that consists of link connectivity for all sites, whereas the equipment level provides connectivity information only for the site at which equipment configuration is taking place. Additionally, the equipment level provides information consisting of which radios each link is connected to and complete equipment connectivity at that particular site. Consequently, link connections hold the special status in that they are the only type of connection which bridges the network and equipment levels of representation.

Several link connection constraints must be satisfied. A site can only have as many links specified in its equipment configuration as there are links connected to it in the network configuration. Link connections (at the equipment level) always start at a radio and end in free space. Only one link may be connected to a radio. Note that there may be more radios than links at the network level, but there will remain extra radios which are not connected to any links. Consequently, these radios would not be part of the equipment connectivity and would constitute an incomplete representation of a network system.

A special type of connectivity specification is supported for configuring DPASes. DPASes only host digroup connections. The function of a DPAS is to interconnect digroups at the channel level of connectivity. DPAS configuration is supported by a DPAS configuration editor window. This window is physically comprised of an input completion pane at the top of the window, with the remaining bottom three-fourths divided column-wise into an input pane on the left and a configuration pane on the right.

3.3.6.4 Circuit Representation

A circuit is a complete path (consisting of connections and equipment) from one MUX-98's input to another MUX-98's input. There is at least one level of multiplexing involved in each circuit. Circuit specification is easily performed by the user, though it has been a very complex task from a design and development perspective.

The addition of circuits should only take place after all sites have been configured at the equipment level. This is because circuit addition requires a complete path from the originating MUX-98 to the destination MUX-98. If an incomplete path is discovered during the path seeking algorithm, then the addition process for that particular circuit is unconditionally aborted and information about the problem area is presented to the user. One side benefit of circuit specification is thus knowledge base consistency checking. Successful completion of the circuit path seeking algorithm indicates that the specified equipment connectivity for that circuit's path is logically sound and meets connectivity constraints imposed upon circuit paths.

3.3.7 Evaluation

The intent of this portion of the effort was to develop a graphical interface tool for the construction of a structural knowledge base representing domain specific knowledge for a communications network system. In its current implementation, GUS is

proving itself to be an effective tool for knowledge base construction. Design criteria and objectives have been satisfied and in some cases exceeded.

Although the present design was directed toward a specific application domain, the design principles employed are domain independent. The motivation behind the development of a generic interface tool for capturing and representing structural knowledge of a variety of application domains is obvious.

The careful attention paid to modularity in designing GUS has resulted in a system which can potentially serve as a prototype for a generic interface tool. Specifically, the use of an objected-oriented approach lends itself to a domain independent extension by allowing the possibility of user-defined object classes. A conceptual perspective of such a generic implementation would require a domain information acquisition tool to be built on top of the design skeleton of GUS. This acquisition tool should permit the very high-level design of an interface with a limited design choice methodology. That is, the construction and tailoring of an interface to a particular application domain should follow a predetermined design process. An obvious design process model is the design process followed for the creation of the application specific tool GUS.

Although the responsibilities of the acquisition tool are numerous, there are significant responsibilities worth noting here. Each step of the design specification process incurred by the acquisition tool should be closer to functional requirements of the application domain. For instance, design could start with conceptual levels of detail and finish with connectivity constraints. The determination of conceptual levels of structural detail of the application domain is a key step which has a significant impact on the interface design. Reflective of the object-oriented approach to design, the next step is recognition of component and connection type objects. Thus, a library of domain specific objects and their attributes should be created and represented. These objects should then be associated with predefined object classes which have generic operation capabilities. In this way, domain specific objects are created and acquire operational capabilities (from an interface perspective) by being associated with a predefined object class. The creation of mouse sensitive regions for domain specific objects and the creation and association of graphic icons and textual commands to these mouse sensitive regions must also be supported. In general, menu types and their options should be reflective of those menus used by GUS. However, high-level creation of certain menu types with user-specified items should be a user-option of the design process.

A limitation encountered with this implementation of GUS is that the window size constrains the size and complexity of a represented network system at both the network and equipment levels. This is a consequence of component objects having a fixed location for display. Investigation of zooming and panning techniques and their potential application to this specific problem would be appropriate. At present, absolute screen pixel coordinates are used for representation *and* display of objects. Zooming could be implemented by using the same absolute representation and a scaling technique on absolute screen coordinates for display. Implementation of panning follows from the scaling technique for zooming. Depending upon the current scale, displayed object coordinates would be relative to a scaled "home" pixel coordinate. Various

regions of the display could then be viewed by detection of mouse cursor movement in editor window margins (similar to scrolling window capabilities). Objects displayed would be displayed relative to the home coordinate whether it is visible or not. The addition of zooming and panning capabilities permits the display of network systems with real world longitude and latitude locations and realistic proportions. Thus, the representation and modeling of existing network systems and the creation and modeling of hypothetical network systems closely related to real world coordinates would be a salient characteristic of the interface tool.

Another point of criticism is configuration completeness. Although consistency checking is provided by connection constraints, it is still possible for a user to construct an incorrect configuration from a completeness perspective. Incomplete configurations are detected when specifying circuits and indicate to the user that the current network structure and corresponding knowledge base must be modified.

3.3.8 Comparison With Existing Tools

An existing knowledge representation tool in use today is Intellicorp's KEE (Knowledge Engineering Environment) system. KEE offers many graphics tools, some of which have many conceptual and functional characteristics similar to those of the graphical interface we have developed.

The KEE system is a development environment for building models and reasoning about and analyzing those models. Within the KEE environment there are graphics tools which help users construct graphic images, image libraries, and interfaces via an object-oriented implementation. Thus, KEE has a frame-based knowledge base consisting of objects and their associated attribute slots. Some of these tools include KEEpictures, ActiveImages, and SimKit. KEEpictures assists the user in constructing customized, graphic images and interfaces. ActiveImages is a library of images constructed with KEEpictures. Of particular interest is the tool SimKit. With SimKit and a library of graphical simulation objects, non-programmers can easily build, run and modify simulations with simple mouse-and-menu commands.

The interesting aspect of SimKit, for our purposes, is not its simulation abilities, but the mechanism and procedure by which models are built and represented. Users are able to interact with the application by manipulating images with a mouse-controlled cursor. A library of icons representing simulation components is used to build complete simulation models. As components are selected from the library's menu of icons for addition to the simulation model, new members of the class of the simulation component represented by the icon are automatically created and added to the model's knowledge base. Attribute slots are utilized to represent the modeled objects' attributes and their corresponding values. Additionally, connections between component models are represented by slots. The salient feature here is that the explicit addition of objects to the knowledge base is avoided by having knowledge base modification be a consequence of graphical editing with the mouse.

In GUS user interaction is primarily through mouse-controlled manipulation of a library of domain specific icons. These icons represent the components and connections

of the application domain. As components are selected from the library of icons for addition, new instances of objects represented by the icon are automatically created and added to the system knowledge base. This is a concept held in common with SimKit. A frame-based knowledge base implementation is also utilized with slots and default values representing modeled physical object attributes. Another major commonality is that knowledge base building is completed implicitly by the addition of component and connection icons to comprise the physical architecture of the target system.

A limitation of our graphical interface is that it is domain specific. The library of domain specific icons is fixed and not modifiable via the interface tool itself. However, this is not a limitation of the graphics capabilities of SimKit. SimKit permits loading in of a library of icons. Custom application libraries can be created with the use of KEEpictures and ActiveImages.

A design goal of our interface, which is not apparent in SimKit, is to have knowledge concerning graphical display of a modeled physical object be loosely coupled to structural knowledge of the system. Although implementation and the extent to which SimKit addresses this goal is unclear, it is believed that our approach is unique. A common approach to coupling graphical display capabilities is via inheritance by mixing in graphical display objects to objects of the application domain. Our approach does not follow this conventional technique, but instead allows the structural knowledge to be represented completely independently from the graphical display knowledge.

3.4 Multistage Negotiation In Distributed Planning

3.4.1 Introduction

We have developed a multistage negotiation protocol that is useful for cooperatively resolving resource allocation conflicts which arise in a distributed network of semi-autonomous problem solving nodes. The primary contributions of such a negotiation protocol are that it makes it possible to detect and to resolve subgoal interactions in a distributed environment with limited communication bandwidth and no single locus of control. Furthermore, it permits a distributed problem solving system to detect when it is operating in an overconstrained situation and act to remedy the situation by reaching a satisficing [17] solution.

Multistage negotiation is specifically *not* intended as a mechanism for goal decomposition in the system, though some goal decomposition is a natural result of negotiation in the context of this paradigm. Our protocol may be viewed as a generalization of the contract net protocol [5, 20, 21]. The contract net was devised as a mechanism for accomplishing task distribution among agents in a distributed problem solving system. Task distribution takes place through a negotiation process involving contractor task announcement followed by bids from competing subcontractors and finally announcement of awards. Multistage negotiation generalizes this protocol by recognizing the need to iteratively exchange inferences drawn by an agent about the impact of its own choice of what local tasks to perform in satisfaction of *global goals*.

Multistage negotiation produces a cooperation strategy similar in character to the Functionally Accurate/Cooperative paradigm [14] in which agents iteratively exchange tentative and high level partial results of their local subtasks. This strategy results in solutions which are incrementally constructed to converge on a set of complete local solutions which are globally consistent. Before describing multistage negotiation in detail, we first motivate the need for a new cooperation paradigm.

3.4.2 Motivation For Multistage Negotiation

The distributed environment in which our negotiation takes place is a network of loosely coupled problem solving agents in which no agent has a complete and accurate view of the state of the network. Problem solving activity is initiated through the instantiation of one or more top level goals at agents in the network. Each top level goal is instantiated locally at an agent and is not necessarily known to other agents. Since the conditions which give rise to goal instantiation may be observed at more than one place in the network, the same goal may be instantiated by two or more agents independently. The desired solution to the problem is any one that satisfies all of the top level goals.

In this type of distributed network, it is very expensive to provide a complete global view to each agent in the system. Communication bandwidth is generally limited. Exchange of enough information to permit each agent to construct and maintain its own accurate global view would be prohibitively expensive. In addition, progress in problem solving would be significantly slower due to a decrease in parallelism attributable to

the need for synchronization in building a complete view. Multistage negotiation has been devised as a paradigm for cooperation among agents attempting to solve a planning problem in this distributed environment. In the remainder of this section, we explain the contributions of multistage negotiation in solving distributed planning problems.

One of the major difficulties which arises in planning systems is detecting the presence of subgoal interactions and determining the impact of those interactions. In distributed applications, the problem is exacerbated because no agent has complete knowledge concerning all goals and subgoals present in the problem solving system. For example, subgoals initiated by one node may interact with other subgoals initiated elsewhere, unknown to the first node. These interactions may become quite complex and may not be visible to any single node in the network. *A key objective of our multistage negotiation is to allow nodes to exchange sufficient information so that these interactions are detected and handled in a reasonable manner.* This objective is achieved by exchanging knowledge about the nonlocal impact of an agent's proposed local action without requiring the exchange of detailed local state information.

Another significant issue that arises in planning is recognizing when goals are not attainable. When satisfaction of a goal requires the commitment of resources, conflicts may arise among goals competing for limited resources. A planning problem is overconstrained if satisfaction of one top level goal precludes the satisfaction of others. Detection of an overconstrained situation in a distributed environment is, again, particularly difficult because no agent is aware of all goals, and each agent has only a limited view of the complete set of conflicts. *When a number of alternative choices for goal satisfaction are known, detection of an overconstrained situation is not possible without either multistage negotiation or a global view.*

In an overconstrained problem, a planning system must reformulate what it seeks as a satisfactory solution. Having several equally important top level goals, the planner must decide which ones should be sacrificed to permit satisfaction of others. Since the distributed network has no agent with sufficient knowledge to serve as an intelligent arbitrator, a consensus must be reached. *Multistage negotiation provides a mechanism for reaching a consensus among those nodes with conflicting goals concerning an acceptable satisficing solution.*

In the following sections, we first describe the problem in more detail, discussing a specific application as an example which illustrates the nature of the planning problem. We then discuss two models of problem solving relevant to this domain: one which is oriented from the perspective of a single goal and one which is node centered. In the fifth section we discuss a multistage negotiation protocol which utilizes these models and has been incorporated in a distributed planner for this problem. We illustrate this protocol with the aid of a simple example. Finally, we discuss ways in which this research extends existing work.

3.4.3 A Specific Application and an Example

In the context of network management and system control for communications systems, a restoral plan consists of a logical sequence of control actions which

allocate scarce resources in order to restore end-to-end user connectivity (circuits). These actions allocate or reallocate equipment and link capacity along some route to specific circuits and are subject to a number of constraints. For example, a circuit is assigned to one of several priority categories. In attempting to restore service, resources belonging to circuits of a lower priority may be preempted. Depending upon the type of circuit, there may be special equipment needs which are not necessarily present at all sites. Available routes through the network may be constrained by lack of certain equipment items such as switches or multiplexers. Thus generation of a restoral plan for a single circuit uses conventional route finding algorithms [23] in combination with knowledge about circuit types and priority, needed equipment, network topology, and equipment configuration at all sites along the restoral path. For any specific circuit there will generally be many alternative restoral plans, so the planning system must then attempt to select a combination of alternatives which restores all circuits.

There are a number of features of this planning problem that make it interesting. There is implicit in this domain the assumption that the knowledge of each agent is incomplete. It may also be inaccurate and inconsistent with that of other agents. Restoral plans must be generated in a distributed fashion because no agent has a global view and reliability issues mitigate against delegating the responsibility for planning to a central node. The overall system goal is one of determining plans for restoral of all interrupted service. Although each agent implicitly knows this goal, it generally will not know all of the specific circuits which require restoral. The planning system need not satisfy the overall goal to be successful. In many instances, the overall goal may be infeasible, and thus a satisfactory plan will fall short of reaching this goal.

The distributed planning problem addressed in this paper and our approach to solving it can best be understood with the aid of an example. A simplified diagram of a small network is shown in Figure 3-6. In this phase of our work, we use a simplified model of a communications system which disregards any constraints arising from equipment configuration at a site. There are five subregions, labeled A, B, C, D, and E, shown. Each site is designated by a letter-number pair, where the letter indicates the subregion in which the site is located. The communication network links are designated by L-number. The control facility for each subregion is located at the site marked with an "*". Each control facility has a planning agent to restore interrupted service. It should be noted that a separate communication network, of substantially lower bandwidth, is not shown, but is assumed to interconnect the control facilities for the purpose of exchanging messages among the agents.

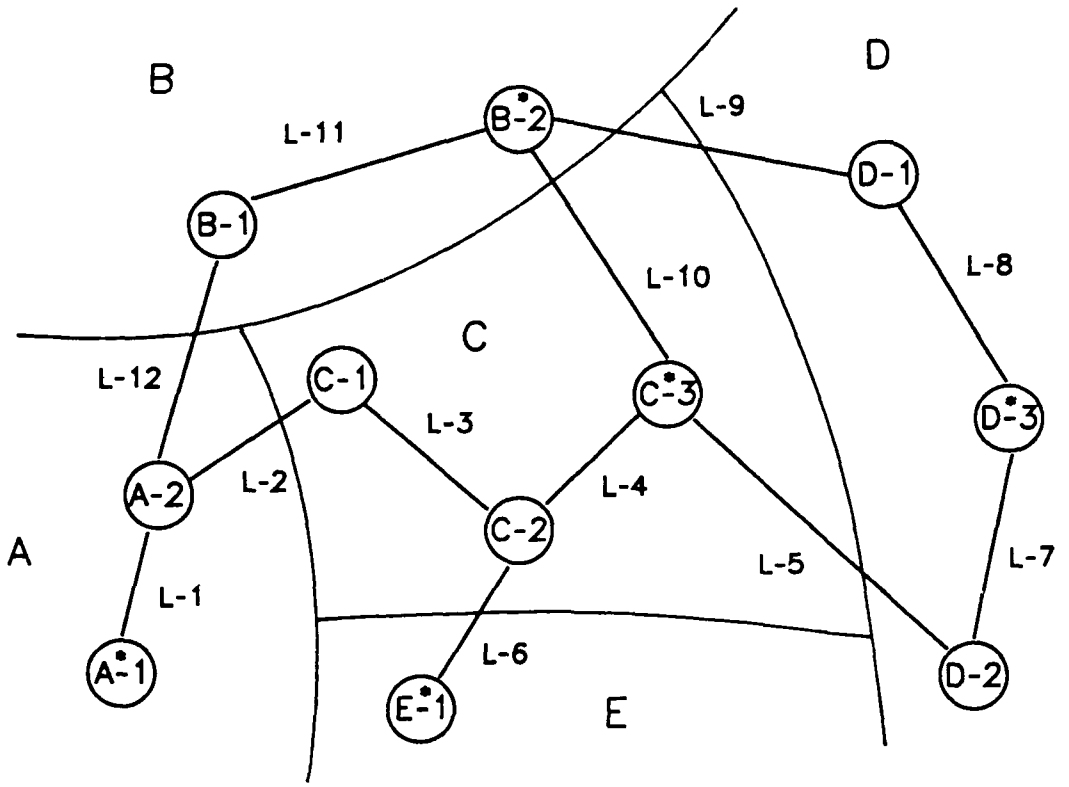


Figure 3-6 Example Network

For the purposes of describing the restoral problem, we assume that there is an equipment malfunction at station B-2 that fails all communication using link L-11. We also assume that each link can handle at most two circuits and that there are four circuits established at the time of the supposed failure. These are described in Table 3-1 by listing the sites and links along the route of each circuit.

ckt-1 (A-1 - :L-1: - A-2 - :L-12: - B-1 - :L-11: - B-2)

ckt-2 (B-1 - :L-11: - B-2 - :L-10: - C-3 - :L-5: - D-2)

ckt-3 (E-1 - :L-6: - C-2 - :L-4: - C-3 - :L-5: - D-2 - :L-7: - D-3)

ckt-4 (B-1 - :L-12: - A-2 :L-2: C-1 :L-3: C-2)

Table 3-1 Circuit Descriptions

To simplify the presentation, these circuits all have the same restoration priority so that none of them should be preferred over the others for restoral in the event of service disruption.

As a result of the presumed failure, two circuits are disrupted, namely ckt-1 and ckt-2 (both use L-11 to get from B-1 to B-2). The planning activity is initiated when an agent observes disruption of a circuit terminating within its subregion and instantiates a restoral goal. In this example, the restoral goals are autonomously instantiated in subregion A (for ckt-1), subregion B (for ckt-1 and ckt-2) and subregion D (for ckt-2). Each agent initially has only the following knowledge about each circuit terminating in its subregion:

(a) a circuit identifier that is unique within the network,
(b) a priority or degree of urgency for restoral,
(c) detailed routing of this circuit within this agent's area of responsibility,
(d) the end stations of the circuit and the agents responsible for them.

In addition, each agent has detailed knowledge concerning the status of resources resident in its subregion.

The first phase of the planning process is plan generation, and since it uses only one stage of negotiation, as in contract nets [5, 20, 21] we shall not consider the details of plan generation here. When viewed from a global perspective, plan generation produces two alternative restoral plans for each circuit. Each plan is represented in Table 3-2 as a list of alternating sites and links, traversing the proposed restoral path.

Plans for goal g1 to restore ckt-1:

g1/p1 (A-1 - :L-1: - A-2 - :L-2: - C-1 - :L-3: - C-2 - :L-4: - C-3
:L-5: - D-2 - :L-7: - D-3 - :L-8: - D-1 - :L-9: - B-2)

g1/p2 (A-1 - :L-1: - A-2 - :L-2: - C-1 - :L-3: - C-2 - :L-4: - C-3
:L-10: - B-2)

Plans for goal g2 to restore ckt-2:

g2/p1 (B-1 - :L-12: - A-2 - :L-2: - C-1 - :L-3: - C-2 - :L-4: - C-3
:L-10: - B-2 - :L-9: - D-1 - :L-8: - D-3 - :L-7: - D-2)

g2/p2 (B-1 - :L-12: - A-2 - :L-2: - C-1 - :L-3: - C-2 - :L-4: - C-3
:L-5: - D-2)

Table 3-2 Alternative Plans

To clarify the example, we have adopted a naming convention for goals and alternative plans which incorporates the circuit name and plan number; thus the two alternative

plans for restoring circuit ckt-1 are designated g1/p1 and g1/p2. It is essential to remember that these are global plans which have been generated in a distributed manner, and no single agent *necessarily* knows of all plans or any one complete plan.

As a result of plan generation, a node produces local alternative plan fragments which may be used to satisfy global goals. Each global plan listed in Table 3-2 is composed of several fragments distributed over a subset of the agents. This is illustrated in Table 3-3.

Goal	Plan Frag.	Resources Used	Cost
g1	1A	L-1, L-2	9
g2	7A	L-2, L-12	9
	Agent A		
g1	2B	L-9	9
	5B	L-10	6
g2	8B	L-9, L-10, L-12	9
	11B	L-12	6
	Agent B		
g1	3C	L-2, L-3, L-4, L-5	9
	6C	L-2, L-3, L-4, L-10	6
g2	9C	L-2, L-3, L-4, L-10	9
	12C	L-2, L-3, L-4, L-5	6
	Agent C		
g1	4D	L-5, L-7, L-8, L-9	9
g2	10D	L-7, L-8, L-9	9
	13D	L-5	6
	Agent D		

Table 3-3 Local Knowledge About Plan Fragments

This table summarizes the knowledge each agent has about goals, alternative plan fragments, and local resources. Plan fragments are numbered and each is identified by a letter indicating the responsible agent. Note that agents are not *explicitly* aware of global alternative plans, but are only aware of local alternatives. For example, even though Agent A has resources needed by both g1/p1 and g1/p2, the local plan fragment is the same in both cases, and thus Agent A "sees" only one alternative plan for goal g1.

This example is considerably oversimplified in order to focus attention on the significant characteristics of this planning problem and to illustrate the cooperation strategy which results from multistage negotiation. The communication network has been simplified so that link capacity is the only resource, and thus there are no constraints arising from local equipment configurations. The number of circuits and link capacities are also much smaller than is typical. Since only two top level goals exist, the

subgoal interactions are simple and can be recognized in only one step. In a more realistic problem, subgoal interactions often involve multiple dependencies and may require several steps of negotiation to detect and resolve.

The features of the planning problem which are important for the discussion of multistage negotiation in this paper are summarized below:

(a) Goals are autonomously generated at nodes in the system.

(b) The same system goal may be generated at more than one node, independently.

(c) Knowledge about local resource availability and potential goal interactions at each node differs from that at other nodes.

(d) Goal satisfaction in general requires nonlocal resources.

(e) The planning problem being addressed is, in general, overconstrained. A choice to satisfy some goals may preclude the satisfaction of others, so choice heuristics are necessary.

(f) Goals are prioritized, but this does not imply a total ordering with respect to priority.

3.4.4 Model Of Problem Solving

The planning problem discussed in the previous section can be viewed in a broader context. In this section we characterize a problem solving model in which multistage negotiation is useful. The search space for a problem of this kind can be considered from two points of view: a task or goal centered perspective and a node centered perspective. Each of these ways of viewing the search space provides a different set of insights with respect to problem solving.

When viewed from the perspective of the system goal, the global problem appears as an AND-OR tree progressing from the system goal (at the root), down through goals and plans, to local plan fragments distributed among the agents. A goal centered view of our example problem is illustrated in Figure 3-7. Two goals have been instantiated, with four alternative plans and several local plan fragments. Of course, since this is a distributed environment, no single agent has a complete view of this tree. Observe that each agent is aware of both goals g1 and g2, but agent D is only aware of one plan fragment for g1, the one which is a component of g1/p1.

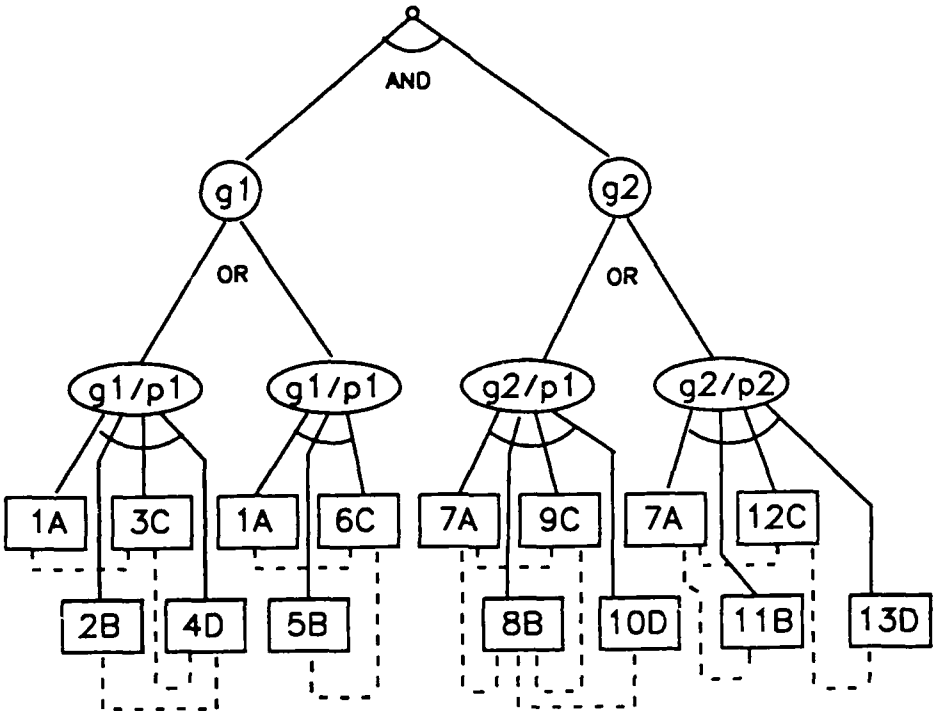


Figure 3-7 Global Search Space

An agent may not simply satisfy a local goal by choosing any plan fragment, but must coordinate its choice so that it is compatible with those of other agents. Formulation of a plan as a conjunction of plan fragments induces a set of compatibility constraints on the local choices an agent makes in satisfaction of global goals. In Figure 3-7 we show the plan fragments interconnected by dashed lines. These dashed lines indicate the local knowledge an agent has about which other agents are involved in compatibility constraint relations with its own plan fragments. Observe that an agent generally does not have *complete* knowledge about these compatibility sets. In our application domain, these constraints involved shared resources between two agents.

From a node centered perspective, plan fragment selection is constrained by local resource availability. An agent cannot choose to execute a set of alternative plan fragments that require more local resources than are available. For example, agent B's local resources permit selection of any pair of its own plan fragments in satisfaction of g1 and g2, whereas agents A, C, and D each can select only one plan fragment. The resulting feasibility tree known to each agent is shown in Figure 3-8. In this figure, resource constraints associated with goals and plan fragments are enclosed by ovals and connected to the appropriate objects with dashed lines. Restoral goals initiated in a subregion are designated with an "*".

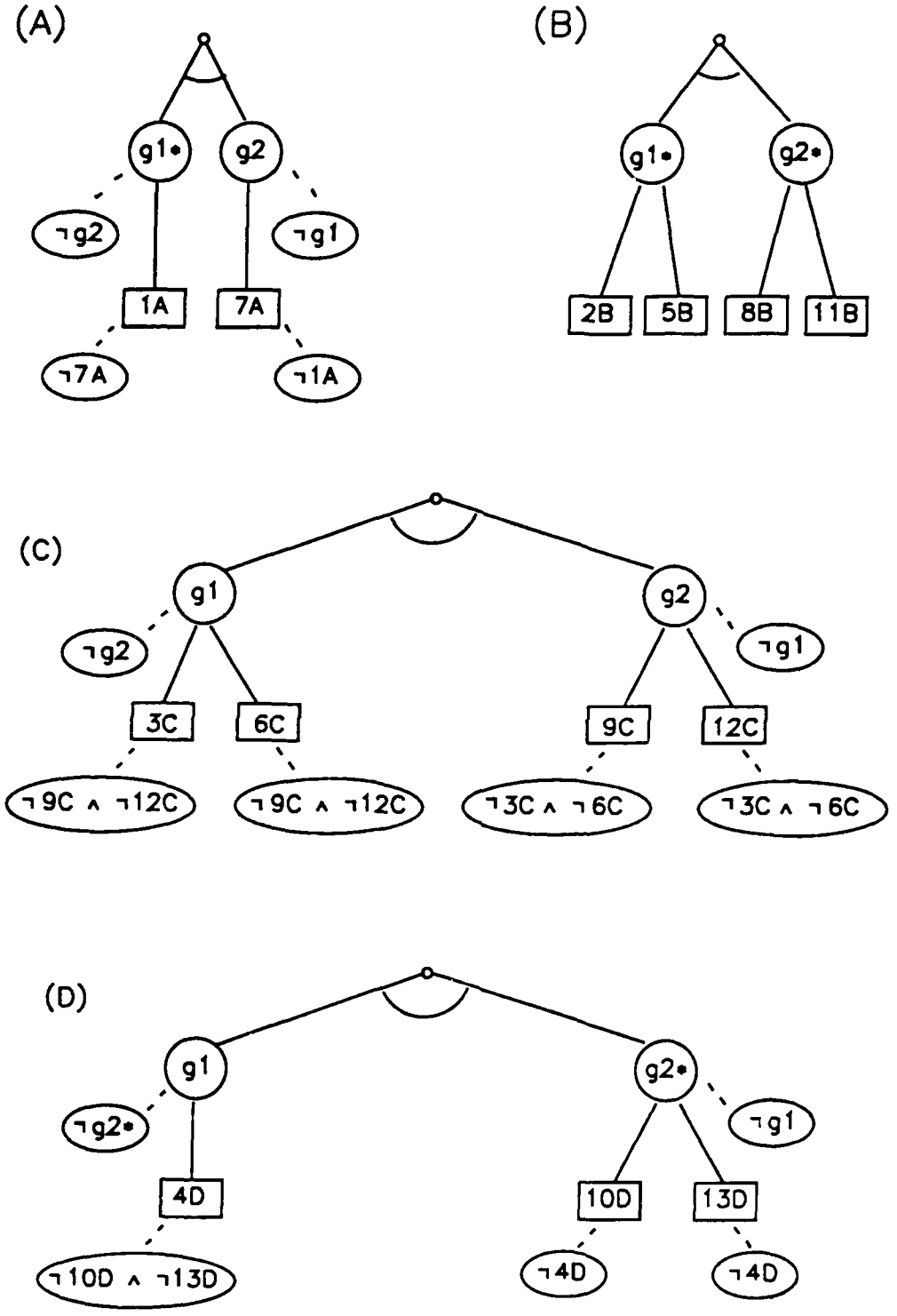


Figure 3-8 Local Feasibility Trees

From each agent's perspective, the search is over a group of alternatives subject to a set of local resource constraints and a set of compatibility constraints imposed by actions of other agents. Multistage negotiation provides a mechanism by which agents coordinate their actions in selecting plans subject to both resource and compatibility constraints. As additional constraints are added to an agent's base of knowledge, its local feasibility tree is augmented to reflect what it has learned.

3.4.5 Multistage Negotiation

In this section, we describe the multistage negotiation protocol we have developed and give an example of its application in the distributed planning problem which has been discussed. We first treat the protocol at a very high level, discussing the general strategy. We then provide more detail as to phases of planning and the role of negotiation in each. The section is concluded with a detailed trace of negotiation and reasoning in each agent pertinent to our simple example.

3.4.5.1 High Level Protocol

Multistage negotiation provides a means by which an agent can acquire enough knowledge to reason about the impact of local activity on nonlocal state and modify its behavior accordingly. When problem solving activity is initiated, agents first engage in a phase of plan generation. Each agent ascertains what alternatives for partial goal satisfaction are locally possible and tenders contracts to appropriate agents for furthering satisfaction of the goals needed to complete these plans. On completion of this phase, a space of alternative plans has been constructed which is distributed among the agents, with each agent only having knowledge about its local plan fragments. An agent then examines the goals it instantiated and makes a tentative commitment to the highest rated feasible set of plan fragments relative to these goals. It subsequently issues requests for confirmation of that commitment to agents who hold the contracts for completion of these plan fragments.

Each agent may receive two kinds of communications from other agents: 1) requests for confirmation of other agents' tentative commitments, and 2) responses concerning the impact of its own proposed commitments on others. Impact of local actions is reported as confirmation that a tentative local choice is a good one or as negative information reflecting nonlocal resource conflict. The agent incorporates this new knowledge into its local feasibility tree. It rerates its own local goals using the new knowledge and possibly retracts its tentative resource commitment in order to make a more informed choice. This process of information exchange continues until a consistent set of choices can be confirmed.

Termination of the negotiation process can be done using system-wide criteria or it can be accomplished in a diffuse manner. If global termination criteria are desired in an application, some form of token passing mechanism [7, 9, 25] can be used to detect that the applicable termination criteria have been met. When synchronized global termination is not required in an application, the negotiation can be terminated by an "irrevocable" commitment of resources. A node initiates plan execution in accordance with its negotiated tentative commitment at some time after it has no pending activities

and no work to do for other agents.

3.4.5.2 Mechanics of Negotiation

When a node begins its planning activity, it has knowledge of a set of top level goals which have been locally instantiated. A space of plans to satisfy each of these goals is formulated during plan generation without regard for any subgoal interaction problems. After plan generation, each node is aware of two kinds of goals: *primary* goals (or p-goals) and *secondary* goals (or s-goals). In our application, p-goals are those instantiated locally by an agent in response to an observed outage of a circuit for which the agent has primary responsibility (because the circuit terminates in the agent's subregion). These are of enhanced importance to this agent because they relate to system goals which *must* be satisfied by this particular agent if they are to be satisfied at all. An agent's s-goals are those which have been instantiated as a result of a contract with some other agent. An agent regards each of its s-goals as a possible alternative to be utilized in satisfaction of some other agent's p-goal.

A plan commitment phase involving multistage negotiation is initiated next. As this phase begins, each node has knowledge about all of the p-goals and s-goals it has instantiated. Relative to each of its goals, it knows a number of alternatives for goal satisfaction. An alternative is comprised of a local plan fragment, points of interaction with other agents (relative to that plan fragment), and a measure of the cost of the alternative (to be used in making heuristic decisions). Negotiation leading to a commitment proceeds along the following lines.

1. Each node examines its own p-goals, making a tentative commitment to the highest rated set of locally feasible plan fragments for p-goals (s-goals are not considered at this point because some other agent has corresponding p-goals).

2. Each node requests that other agents attempt to confirm a plan choice consistent with its commitment. Note that an agent need only communicate with agents who can provide input relevant to this tentative commitment.

3. A node examines its incoming message queue for communications from other nodes. Requests for confirmation of other agents' tentative commitments are handled by adding the relevant s-goals to a set of active goals. Responses to this agent's own requests are incorporated in the local feasibility tree and used as additional knowledge in making revisions to its tentative commitment.

4. The set of *active goals* consists of all the local p-goals together with those s-goals that have been added (in step 3). The agent rates the alternatives associated with active goals based on their cost, any confirming evidence that the alternative is a good choice, any negative evidence in the form of nonlocal conflict information, and the importance of the goal (p-goal, s-goal, etc.). A revised tentative commitment is made to a highest rated set of locally consistent alternatives for active goals. In general, this may involve decisions to *add* plan fragments to the tentative commitment and to *delete* plan fragments from the old tentative commitment. Messages reflecting any changes in

the tentative commitment and perceived conflicts with that commitment are transmitted to the appropriate agents.

5. The incoming message queue is examined again and activity proceeds as described above (from step 3). The process of aggregating knowledge about nonlocal conflicts continues until a node is aware of all conflicts in which its plan fragments are a contributing factor.

Two issues need clarification at this point. One deals with the question of termination and the other is concerned with the quality of the result obtained through negotiation (relative to optimality).

Negotiation in this framework continues as long as there are any pending activities in an agent. The only way a situation leading to nontermination could arise involves an agent's making a tentative commitment and subsequently entering a cycle of retracting and remaking that commitment indefinitely. It is not reasonable to expect that an agent should never retract a tentative commitment. It is also not reasonable to expect that an agent would never decide, based on new knowledge, to recommit to an alternative it had previously rejected. An agent's local reasoning must be able to detect when it is making a tentative commitment it has previously made with no new knowledge. Negotiation activity in an agent terminates either when it has no pending activity and no incoming communications or if an attempt is made to return to a previous commitment with no new knowledge from other agents. Endless loops of commitment and decommitment are prevented through this mechanism.

The other issue of importance at this point is related to the quality of the result obtained through negotiation. In the initial negotiation stage, each agent examines only its p-goals and makes a tentative commitment to a locally feasible set of plan fragments in partial satisfaction of those goals. Since each agent is considering just its p-goals at this stage, the only reason for an agent's electing not to attempt satisfaction of some top level goal is that two or more of these goals are locally known to be infeasible. (This corresponds to an overconstrained problem.)

In subsequent stages of negotiation, both p-goals and relevant s-goals are considered in making new tentative commitments. The reasoning strategy employed at each agent will only decide to forego commitment to one of its p-goals if it has learned that satisfaction of this p-goal precludes the satisfaction of one or more other p-goals elsewhere in the system. *If the system goal of satisfying all of the p-goals* instantiated *by agents in the network is feasible,* ***no agent will ever be forced to forego satisfaction of one of its p-goals*** *(because no agent will ever learn that its p-goal precludes others), and a desired solution will be found.* If, on the other hand, the problem is overconstrained, some set of p-goals cannot be satisfied and the system tries to satisfy as many as it can. While there is no guarantee of optimality, the heuristics employed should ensure that a reasonably thorough search is made.

To make these concepts concrete, multistage negotiation is applied to the simplified planning problem discussed in the previous sections.

3.4.5.3 Example

We return to our example of planning activity, assuming that each agent has the knowledge depicted in the appropriate part of Figure 3-8. A summary of the transactions that occur during negotiation to achieve plan selection is shown in Table 3-4. This table is segmented by agent and by "time slice" to convey a sense of progress in problem solving through negotiation. The notational conventions are relatively simple. Tentative commitment to a locally known activity and the associated communication issued to an appropriate agent is denoted in the form (plan fragment name; message -> agent). Exchange of conflict information is indicated in the form (conflict; type of conflict -> agent). To make the trace easier to follow, each received message is noted in the form (source agent -> message).

As is evident in Table 3-4 negotiation begins with tentative commitments to alternatives in agents A, B, and D. Though the problem is overconstrained (it is not possible to restore both ckt-1 and ckt-2), no agent is yet aware of that fact. In response to the initial tentative commitments, there is activity in agents A and C. Agent A knows that it cannot act to satisfy both g1 and g2, but it does *not* know if this precludes satisfaction of g2 (since g2 is an s-goal, there might exist another global plan not requiring any action by A). Since A recognizes the need to *attempt* satisfaction of its own p-goal first, agent A informs agent B there is a conflict between what B requested and satisfaction of one of A's p-goals. Thus A has given B the knowledge that the plan fragment B selected would force A to forego one of its p-goals.

Agent C has now received three communications requesting that plan fragments be extended. It observes that it can effect a plan completion for g1, satisfying both the request from A and the request from B. It also observes that it cannot satisfy both g1 and g2 with use of its locally known plan fragments due to local resource constraints. Since it has the opportunity to complete a plan for ckt-1 and not for ckt-2, it elects to tentatively commit its resources to plan fragment 6C. Messages reflecting this commitment are formulated and transmitted to A and B, while a message indicating the conflict in C is sent to D.

As a result of this second round of communications, activity in subregions B and D is concerned with exploring the remaining alternatives they have for restoral of ckt-2. An acceptable plan for ckt-1 is already reflected in tentative commitments. Agent B elects to try plan fragment 8B and agent D elects to try 10D. Agent B learns that an attempt to satisfy g2 via 8B also fails in A, so it now knows that the problem is overconstrained. Based on the fact that a way of satisfying g1 has already been located, B elects to forego satisfaction of g2 and advises D that it should also give up on g2. Negotiation terminates with tentative commitments reflecting a plan choice for g1.

Agent A	Agent B	Agent C	Agent D
1A; OK? -> C	11B; OK? -> A 5B; OK? -> C		13D; OK? -> C
B -> OK? 11B conflict; (11B AND *not* p-goal g1) -> B		A -> OK? 1A B -> OK? 5B D -> OK? 13D match 6C with 1A and 5B 1A is OK -> A 5B is OK -> B conflict; (13D AND *not* g1 via C) -> D	
C -> 1A is OK	A -> (11B AND *not* p-goal g1) C -> 5B is OK 8B; OK? -> A		C -> (13D AND *not* g1 via C) 10D; OK? -> B
B -> OK? 8B conflict; (8B AND *not* p-goal g1) -> B	D -> OK? 10D 8B; OK? -> C		
	A -> (8B AND *not* p-goal g1) B knows g1 and g2 not both possible (*not* both g1 and g2) -> D	B -> OK? 8B conflict; (8B AND *not* g1 via C) -> B	
	C -> (8B AND *not* g1 via C)		B -> (*not* both g1 and g2)

Table 3-4 Summary of Transactions During Negotiation

In concluding this section we summarize, by "time slice", changes to the local feasibility trees that take place during the negotiation illustrated in Table 3-4.

Slice 1:
No changes

Slice 2:
No changes in constraints by A
6C is tentatively committed to a *complete* plan by C

Slice 3:
1A is marked as tentatively satisfying g1 by A
5B is marked as tentatively satisfying g1 by B
Agent B adds the constraint (*not* g1) to 11B
Agent D adds the constraint (*not* g1 via C) to 13D (Note that in this example, the new constraint on 13D is, in fact, redundant. In other examples, with a more complex set of goals, new constraints propagated in this way often provide additional information.)

Slice 4:
No changes

Slice 5:
Agent B adds the constraint (*not* g1) to 8B
Agent B propagates the constraint (*not* g1) on 8B and 11B to their parent, g2.
Agent B now knows the problem is overconstrained.

Slice 6:
Agent D modifies the constraint (*not* g1) on goal g2* to (*not* g1*).
Agent D now knows the problem is overconstrained.

This example illustrates ways in which knowledge is integrated into the local feasibility tree as it is acquired through negotiation. It shows how knowledge aggregated at the level of plan segments can be propagated in drawing inferences concerning interactions at the goal level. It also shows how the network of agents can become aware that it has an overconstrained problem.

3.4.6 Concluding Remarks

We have developed a new paradigm for cooperation in distributed problem solving systems. This paradigm incorporates features found in two cooperation strategies treated in the literature: the contract net protocol [5, 20, 21] and the FA/C paradigm [14]. It has been devised to permit an agent in a distributed problem solving system to acquire enough knowledge to reason about the impact of local activity on nonlocal state and to modify its behavior accordingly.

Three characteristics of distributed planning problems motivate development of a more general cooperation paradigm. First, subgoal interaction problems that arise in the context of a distributed planning system in which agents do not have a global view are very difficult to detect and even more difficult to handle in a reasonable way. Second, many application domains embody planning problems that are overconstrained. When these planning problems are addressed by a network of planning agents, it is essential that the system be able to determine whether or not the problem is overconstrained. Third, when the planning problem is overconstrained, it is necessary for the agents involved to arrive at an agreement as to a set of goals whose satisfaction is regarded as an acceptable solution to the problem at hand. None of these issues can be resolved in the context of the previously proposed cooperation paradigms without the exchange of sufficient knowledge as to permit each agent to construct a global view.

Another factor motivating formulation of a more general cooperation paradigm is the observation that many application domains have characteristics that distinguish them from other multi-agent planning problems which have been investigated. The strategies suggested by Lansky [13] and Georgoff [11] dealing with planning for a multiple agent domain by a centralized planner are not applicable in situations where there is no central planner. In addition, the agents in our networks are not motivated purely by self interest. They are interested in cooperating to achieve some goals pertinent to system performance. For this reason, the metaphor proposed by Genesereth and others [10] does not represent the domain characteristics. It should be noted, however, that our metaphor can be adapted for use in networks of agents which are selfish (as long as they do not lie a great deal).

The mechanisms presented in this paper are related to the techniques that have been utilized in conventional planning systems. Each agent in our system builds a data structure analogous to the Table of Multiple Effects used by NOAH [18] and NONLIN [24] in detecting subgoal interactions. This structure is incrementally built using knowledge gleaned through negotiation. In detecting and resolving conflicts, a form of criticism analogous to that performed by NOAH's Resolve Conflicts critic is employed. Criticism is necessary in our distributed problem solving systems for the same reason it was needed in NOAH - decisions are made initially based on local criteria, whereas nonlocal conditions affect the viability of those decisions. Unlike NOAH (and like NONLIN), alternatives are not discarded after they have been rejected. Backtracking in the form of revised tentative commitment is a feature of the protocol.

In many planning problems, the constraints arising from resource availability are very important in determining a satisfactory solution to the planning problem. We have found that resource constraints play a crucial role in our system as well. The ability to reason about resources is critical in determining adequate solutions. This was recognized in the design of SIPE [26]. Since we have no central planner, the mechanisms for reasoning about resources are somewhat different from those employed in SIPE, but resources as a factor in problem solving are just as important to multistage negotiation as they were in SIPE.

BIBLIOGRAPHY

[1] D. G. Bobrow, "Qualitative Reasoning About Physical Systems: An Introduction," **Artificial Intelligence, vol. 24,** December 1984.

[2] H. Brown, C. Tonge, and G. Foyster, "Palladio: An Exploratory Environment for Circuit Design," **IEEE Computer,** December 1983, pp. 41-56.

[3] S. E. Conry, R. A. Meyer, and V. R. Lesser, "Multistage Negotiation in Distributed Planning", University of Massachusetts, Amherst Massachusetts 01003, COINS Technical Report 86-67 December 1986.

[4] S. E. Conry, R. A. Meyer, and J. E. Searleman, "A Shared Knowledge Base for Independent Problem Solving Agents," **Proceedings of the Expert Systems in Government Symposium,** IEEE Computer Society, McLean, Virginia, October 1985.

[5] R. Davis and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving," **Artificial Intelligence, vol. 20, no. 1,** January 1983, pp. 63-109.

[6] B. Delagi, "Care User's Manual," Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 1986.

[7] E. W. Dijkstra and C. S. Scholten, "Termination Detection for Diffusing Computations," **Information Processing Letters, vol. 11, no. 1,** August 1980, pp. 1-4.

[8] R. Fikes and T. Kehler, "The Role of Frame Based Representation in Reasoning," **Communications of the ACM, vol. 28, no. 9,** September 1985, pp. 904-920.

[9] N.Francez,"DistributedTermination",**ACMTransactionsonProgrammingLanguages and Systems, vol. 2, no. 1,** January 1980, pp. 42-55.

[10] M. R. Genesereth, M. L. Ginsberg, and J. S. Rosenschein, "Cooperation Without Communication," **Proceedings of the National Conference on Artificial Intelligence (AAAI-86),** August 1986, pp. 51-57.

[11] M. Georgeoff, "Communication and Interaction in Multi-Agent Planning," **Proceedings of the National Conference on Artificial Intelligence (AAAI-84),** August 1984, pp. 125-129.

[12] C. Hewitt, "Concurrency in Intelligent Systems." **AI Expert (Premier),** 1986, pp. 44-50.

[13] A. L. Lansky, "Behavioral Specification and Planning for Multiagent Domains," Technical Note 360, SRI International, Menlo Park, CA, November 1985.

[14] V. R. Lesser and D. D. Corkill, "Functionally Accurate, Cooperative Distributed Systems," **IEEE Transactions on Systems, Man,** and **Cybernetics, vol. SMC-11, no. 1,**

January 1981, pp. 81-96.

[15] D. J. MacIntosh and S. E. Conry, "SIMULACT, A Generic Tool for Simulating Distributed Systems, **Tools for the Simulation Profession,** The Society for Computer Simulation. San Diego, CA (Orlando, Florida, April 1987), pp. 18-23.

[16] M. C. Maguire, "A Review of Human Factors Guidelines and Techniques for the Design of Graphical Human-Computer Interfaces," **Computers and Graphics, vol. 9, no. 3,** 1985, pp. 221-235.

[17] J. G. March and H. A. Simon, **Organizations,** Wiley, 1958.

[18] E. D. Sacerdoti, **A Structure for Plans and Behavior,** Elsevier - North Holland, New York, 1977.

[19] E.Schoen,"TheCAOSSystem,"KnowledgeSystemsLaboratory,DepartmentofComputer Science, Stanford University, Report no. KSL-86-22, March 1986.

[20] R. G. Smith, "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver," **IEEE Transactions on Systems, Man,** and **Cybernetics, vol. SMC-10, no. 12,** December 1980.

[21] R. G. Smith and R. Davis, "Frameworks for Cooperation in Distributed Problem Solving,"**IEEE Transactions on Systems, Man,** and **Cybernetics, vol. SMC-11, no. 1,** January 1981, pp. 61-70.

[22] G. Steele, **Common LISP,** Digital Press, Burlington, MA., 1984.

[23] A. S. Tanenbaum, **Computer Networks,** Prentice-Hall, 1981.

[24] A. Tate, "Project Planning using a Hierarchic Non-Linear Planner," Research Report No. 25, Department of Artificial Intelligence, University of Edinburgh, August 1976.

[25] S. Vinter, K. Ramamritham, and D. Stemple, "Recoverable Communicating Actions in Gutenberg," **Proceedings of the International Conference on Distributed Computing Systems,** May 1986.

[26] D. E. Wilkins, Domain-Independent Planning: Representation and Plan Generation," **Artificial Intelligence, vol. 22,** pp. 269-301.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

DATE

FILMED

DTIC

10-88